22883
PATENT TRADEMARK OFFICE
PTO Customer Number 22883

Title:	High-Speed Low-Power CAM-based Search Engine
Inventor:	Abdollahi-Alibeik
Docket:	204.1001.02

TECHNICAL APPENDIX

37 Total Pages

(not including this cover page)

COVER PAGE ONLY – NOT PART OF TECHNICAL APPENDIX

[NO PAGE NUMBER]

am

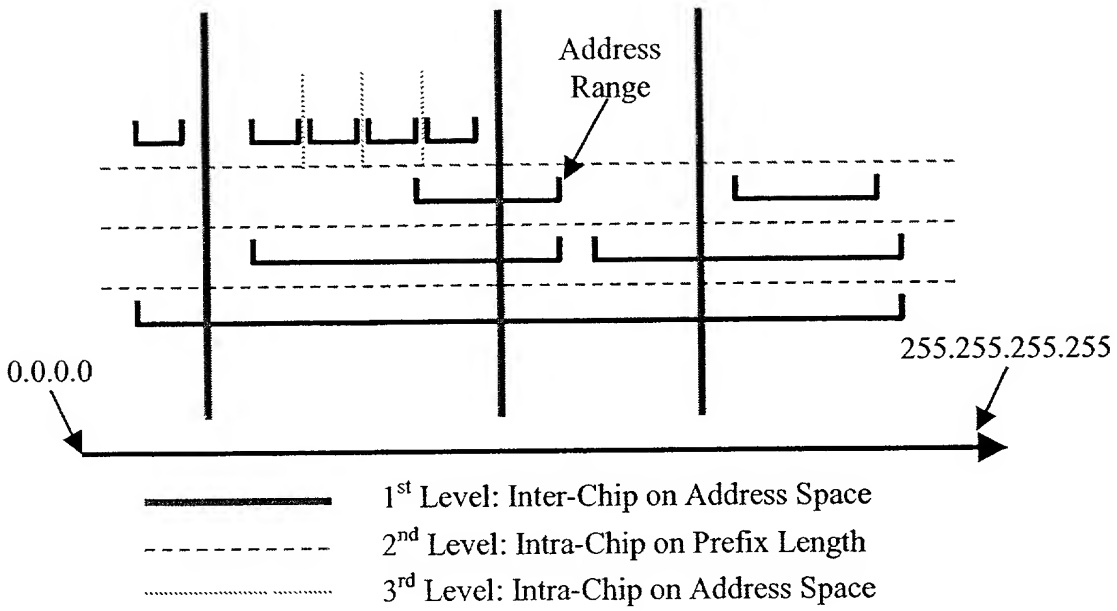


Figure 1: Conceptual overview of the architecture for the lookup method

1.0 Overview

One of the limiting factor in future high-speed IP routers is the problem of performing the longest prefix match to decide out of which port the incoming packet should be routed. The longest prefix match involves looking at the destination address on the incoming packet and finding the longest prefix in the routing table that matches it. As the rate at which data packet can be transmitted over the optical fibers is increasing, the number of lookups that need to be done to keep with this speed is also increasing rapidly. Here we propose a hardware solution in the form of a CAM (Content Addressable Memory) based ASIC (Application Specific Integrated Circuit) that allows lookups to keep up with transmission speeds over optical fibers for the next several years. The advantages of this method are:

- It does not suffer from the high power requirements of usual CAM implementations allowing the use of cheaper packaging and higher density reducing the chip count. Power does not scale with increasing table size, unlike conventional implementations.
- It allows the use of a binary CAM structure in place of a ternary CAM (which can store don't cares) giving higher table entries per chip.

- It has low latency which is beneficial to applications like real time voice and video transmission
- It can support a high lookup rate allowing the routing of a large amount of traffic
- It allows several chips to be operated in parallel with ease, to support large lookup table sizes as there is no communication required between chips to decide the best match

This method is easily extendable to solving other problems which require looking an entry up in a table at a high rate.

2.0 Architecture Description

Each entry in the lookup table consists of a prefix of a certain length. For example, a 32 bit address with a prefix length of 16 is 23.123.0.0/16. This prefix can be thought to represent a range of addresses from 23.123.0.0 to 23.123.255.255. In figure 1 each of these ranges is represented by a square bracket facing upwards. Entries of the same prefix length are placed on the same line.

Searching entries takes power roughly proportional to the number of entries that need to be searched. This scheme saves power by searching only a few entries out of the entire table. The way the table is divided is shown in figure 1. Depending on the technology, chip size and table size, several chips may be required to save and search the entire table. Hence, the first division is between chips. Each chip contains entries from only a certain range of the address space. Entries that cross this boundary are put in both chips. Thus depending on the range in which the address to be matched falls, only one chip needs to be searched for it.

Within each chip the entries are divided into several packs. These packs will be referred to as *banks*. Each of these banks shares a mask entry, which stores the information on the significant bits in the prefix. This allows the entries to be stored in smaller binary CAMs instead of ternary CAMs, which are otherwise required. Since, each bank contains entries of the same length the entries cannot overlap with each other. Thus, each address will get at the most one match. This eliminates the need to have a priority encoder within each bank to resolve multiple matches. For these reasons the second division is based on prefix length.

The third division is from bank to bank. Depending on the number of entries in each prefix length on each chip several banks may be required to store these entries. Each bank

contains entries contained in a particular address range. Each address lookup needs to only activate one of these banks per prefix length, further reducing the power requirement. A priority encoder is required between banks to determine which was the longest prefix match among the matches from different prefix lengths.

Note that depending on the specific application, technology used and table size, the number, order or type of this division can be changed to obtain the optimal design.

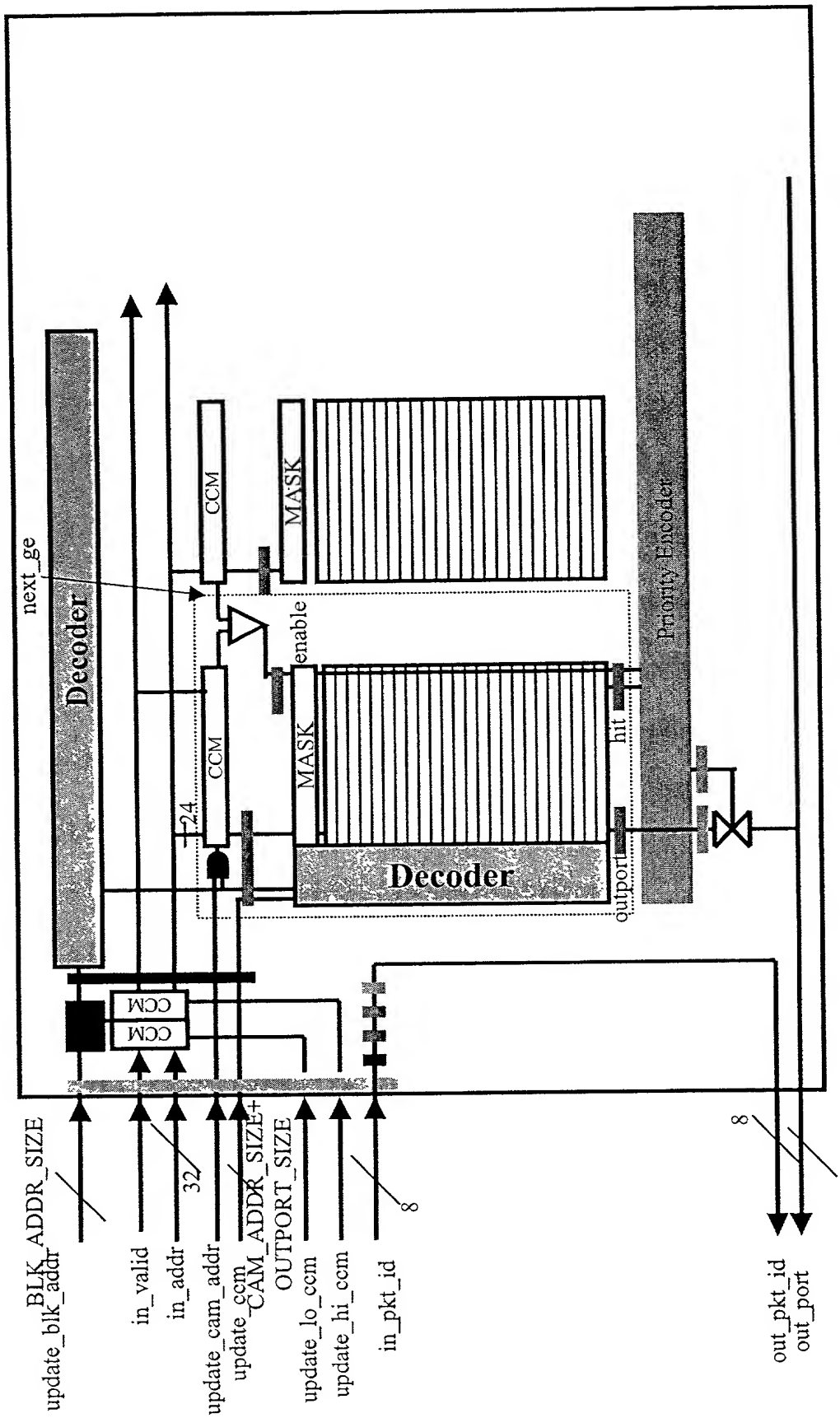


Figure 2: Schematic of chip used in the implementation of the lookup method

3.0 Functional Block Level Implementation

This section deals with the functional block level implementation, while the details of circuit implementation is presented in the following sections. Figure 2 shows the schematic of the implementation. Each thick solid line represents a flipflop. Thus, the regions between flipflops of the same color lie in the same clock domain. The functioning of this schematic will be explained by going through a lookup (an address) and add/delete prefix cycle.

Lookup:

A particular interface is assumed here for the sake of discussion. In a cycle in which there is an address to be looked up, the address is put on the **in_addr** bus, the packet ID is put on the **in_pkt_id** bus and **in_valid** is asserted. Next this address has to go through the first check to find out if it is in the same range as the address in this chip. This is the search on the first division. This search is done by the use of CCM (Content Comparing Memory). In this implementation, without loss of generality, CCM is used to compare the incoming data to that in the memory and computes if it is greater than or equal to the one in the memory. A possible implementation of the CCM is presented in the next sections.

So, in the next cycle the incoming address is compared against two CCMs to check if it is in the right range. The CCM contain the maximum and minimum of the range of address contained in that chip. Chips that do not have addresses in the right range do not have to do any further work on this address saving power. The chips that does match now passes on the address to the CAM banks in the next cycle.

Now, as mentioned before each of these CAM banks contain entries with the same prefix length. This prefix length is encoded in the mask present in each bank. The data in the mask decides which bits of the incoming address will be compared with the entries in the bank. Each CAM bank also contains a CCM. This CCM stores and compares the least possible address that will match the entries in the table with the incoming address. If the incoming address is found to be greater than or equal to the data in the CCM but less than (i.e. not greater than or equal to) the one in the next bank which contains addresses of the same prefix length, then and only then the incoming address is passed to the rest of the CAM bank for comparison. This requires CAM banks with prefixes of the same length to be placed next to each other and the addresses to be sorted between the banks. Note that the addresses within a bank need not be sorted as only one

match can be made for entries of same prefix length. The last in a chain of CAM banks with same prefix length should not compare the incoming address with the next CAM bank (as that contains prefixes of different length). This is achieved by introducing the **last** bit. So for the last CAM bank in a chain (which has the **last** bit set) comparison is carried out only with one CCM.

In the next cycle the comparison within each CAM bank that matched (at the most one per prefix length) is carried out. The circuit operation and design of these CAM cells is detailed in the following sections and hence, will not be covered here. It is sufficient to say here that each row of CAM cells (which contain one entry) have an associated memory row (e.g. SRAM) containing the tag (which could be the port address that the packet needs to leave the router by). If a match is found between the incoming address and one of entries in the bank, corresponding tag is outputted and a **hit** line is asserted.

In the next cycle the priority encoder decides which of the CAM banks has got the longest prefix match. Again, the workings of the priority encoder are explained in detail in the following sections. The priority encoder decides the CAM banks with the highest priority and lets it output its tag (which is the longest prefix match) onto the **out_port** bus.

Update:

This section shall detail how the data structure is maintained. A processor that maintains the update engine gives the update commands. To allow lookups to take place without being held up by updates, each update command maintains the data structure intact. This requires all the CCMs and CAMs at various levels to be updated in one pipelined operation (so as to leave the data structure ready to do a lookup in the next cycle). This means that each update is one clock cycle long and updates each section as it travels down the pipeline. The lookup operation can resume after the clock cycle in which the update is introduced to the pipeline.

To add a new entry to a chip, the entry is placed on the **in_addr** bus and the corresponding tag is placed on the **in_port** bus and the **packet_update** is asserted. The bank address that this update is directed to is put on the **update_blk_addr** bus, while the row number within this bank is put on the **update_cam_addr** bus. Now, this addition might change the data structure, so as to require the modification of the following CCMs:

- Bank CCM: If the entry is the smallest in that bank, the CCM content has to be updated. The **update_ccm** bus is asserted which ensures this. Note that the **in_addr** should contain the

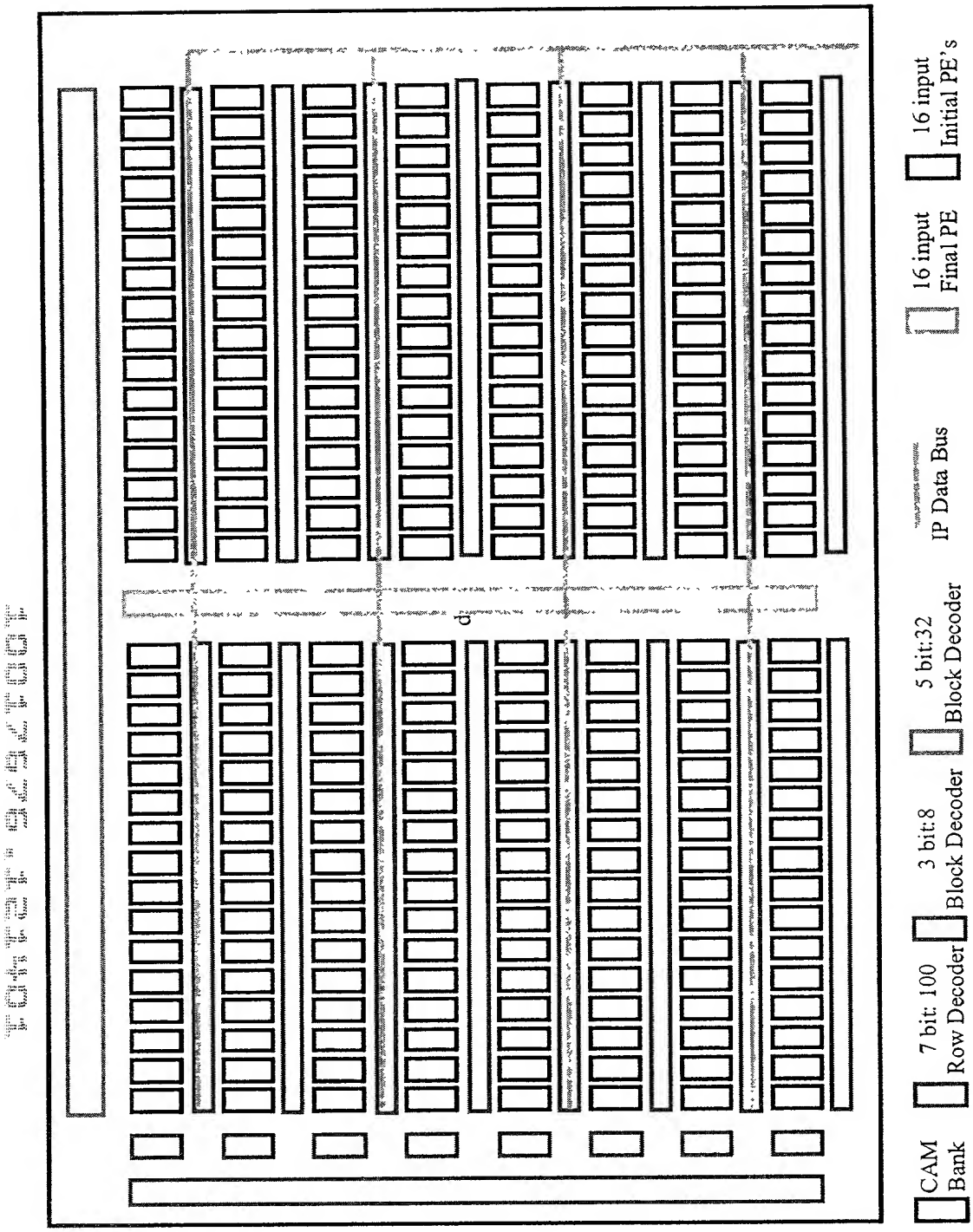


Figure 3 : Floor Plan of the Chip

technology). Thus we obtain a very low power design, since for each lookup only one chip is activated and in each chip at most 25 CAM banks are fired.

smallest address that matches the new entry. The mask in the CAM bank will ensure that the relevant bits are ignored during lookup.

- Lo_CCM: This contains the lowest address than can get a match on this chip. Thus, if the incoming entry is the smallest in the chip, the **update_lo_ccm** is asserted. Again the **in_addr** bus should contain the smallest address that matches the new entry.
- Hi_CCM: This contains the highest address that can get a match on this chip. Thus, if the incoming entry is the largest in the chip, the **update_hi_ccm** is asserted. In this case the **in_addr** bus should contain the largest address that matches the new entry. Note that this update will never require the concurrent updating of the bank CCM. So, putting the largest address on the in_addr bus will not cause a problem.

A delete is similar to an add, except that the entry is set to a special value that will never match a valid incoming address. In the design part we first came up with a compact circuit implementation of the CCM cells. Then we tried to optimize the critical path delay to obtain the maximum speed.

4.0 Architecture and Floorplan

One of the issues with the architecture we chose was the update. At first glance it seems that the update will be very time consuming and of the order of number of entries. But we observed that although the IP entries should be sorted between the CAM banks, they do not have to be sorted inside each CAM bank. This reduces the number of operations needed for each update from O(N) (N = number of entries) to O(N/M) in which M is the number of rows in each CAM bank. In our architecture we assumed M=100, so the update operation is on the order of 1000 operations. In rare cases when the boundary between prefix lengths should be moved across the CAM banks, the number of operations for update may increase. By assigning enough number of CAM banks to each prefix length based on statistics we try to avoid these kinds of updates as much as possible.

The other issue with our architecture is that we have to have extra bins in each chip to take care of the storage area which is wasted between prefix length boundaries. We need to have 25 extra bins (number of different prefix lengths, from 8 to 32) per chip. By looking at the floor plan in fig. 3, the area penalty is only 10%. This area penalty will decrease as we go to larger chips and smaller feature size technologies (less than 2% for a $1cm^2$ chip size in 0.15μm

The floorplan as mentioned above is shown in fig. 3. Also the pipelining of the operations is shown in fig. 2. All latches with the same color represent one operation cycle. The latency for each chip (and for the whole lookup since one chip per lookup is activated) is 6. The expansion of the system by adding chips is also very simple and is only limited by the number of address lines allocated for addressing the memory locations.

For estimating the area, we used the formula given in the project handout. For CAM cells we assumed an area of 50λ×40λ., since we are using smaller transistors than the standard cell and we could stack the vias. For SRAM cell we assumed an area of 28λ×50λ, since the SRAM height is the same as the CAM cell height. This way each CAM bank will have an area of 5.7 $M\lambda^2$. The logic for each bank requires 1.3 $M\lambda^2$. Our Priority Encoders require 4 $M\lambda^2$. Our decoders require 60 $M\lambda^2$. So the total chip area will be 0.74 cm^2. Since the aspect ratio of our chip is 1.11:1, the chip will be 0.91cm×0.82cm.

4.1 Functional testing and Verilog Status

The objective behind writing the verilog code was

- Test the soundness of the idea
- Check for any potential architectural bottlenecks (like large number of wires running all over the chip)
- Ensure the update can be implemented keeping without impacting the lookup my much –

We have met these objectives. The verilog code was written right down to the functional block level (i.e. just above the gate level). For example we implemented each CAM block with all the associated periphery logic. We could not figure out how to do iterative wiring and vectorized instantiations in verilog and since we wanted to implement the model with correct functionality we took recourse to Perl to do the needful.

The controller was implemented mostly procedurally. While lookup was properly implemented, update is much more difficult. So we just implemented a simple update algorithm that cannot take care of spills to adjacent bins. However, we did write a Perl script that took the *lookup_table* and created the initialization file with correct bin partitions.

Since we have a three hash structure we need to worry about three kinds of hash overflows. For overflows in the address space hashes the update algorithm is O(Number of Bin~2000) for overflows in prefix length hashes the update algorithm is O(Number of entries~200K). We devised the architecture in such a way that any CAM and the affected CCMs can be written in the same clock cycle without destroying the data structure integrity. Thus we can continue doing lookups, using only the idle cycles to do the updates. For address space hash overflows this will entail a penalty of O(1%) which is quite small and can be sliped in with holding up the lookup at all. However a prefix length hash overflow can in the worst case hold up lookup for milliseconds. The saving grace is that these updates (i.e. changes in distribution of routing prefix length) are likely to be very infrequent O(months) and thus with clever algorithms will keep this penalty down to a minimum.

4.2 Setup of the circuit critical paths

From the figure of the overall chip operation in section 2.0, there are 6 cycles per chip. The first and last cycles are I/O. The second and third cycles are CCM operations and the fourth one is the CAM bank operation. The fifth one is the chip priority encoder. The CCM operations and CAM operation are very similar to each other and both seem to be in the critical path. So both of these circuits should be simulated. The priority encoder is a static logic evaluation and should not be a problem in terms of timing, but since it is an important function of the chip, it was simulated.

The placement of the wires for the units are shown in fig. 4.

5.0 Circuit Approach and Simulation Method

As we discussed in the previous sections, we have 2 basic Content Comparable Memories: The usual CAM and the CCM. For the usual CAM we used the series matchline structure (or the NAND match chain). As mentioned before this was used to save power. Since we have a large number of CCM also (2048+ of them), we had to come up with a compact structure for this memory. We observed that for comparing our IP address with the CCM content, we can subtract these 2 numbers and see if the result is a negative number or not. In

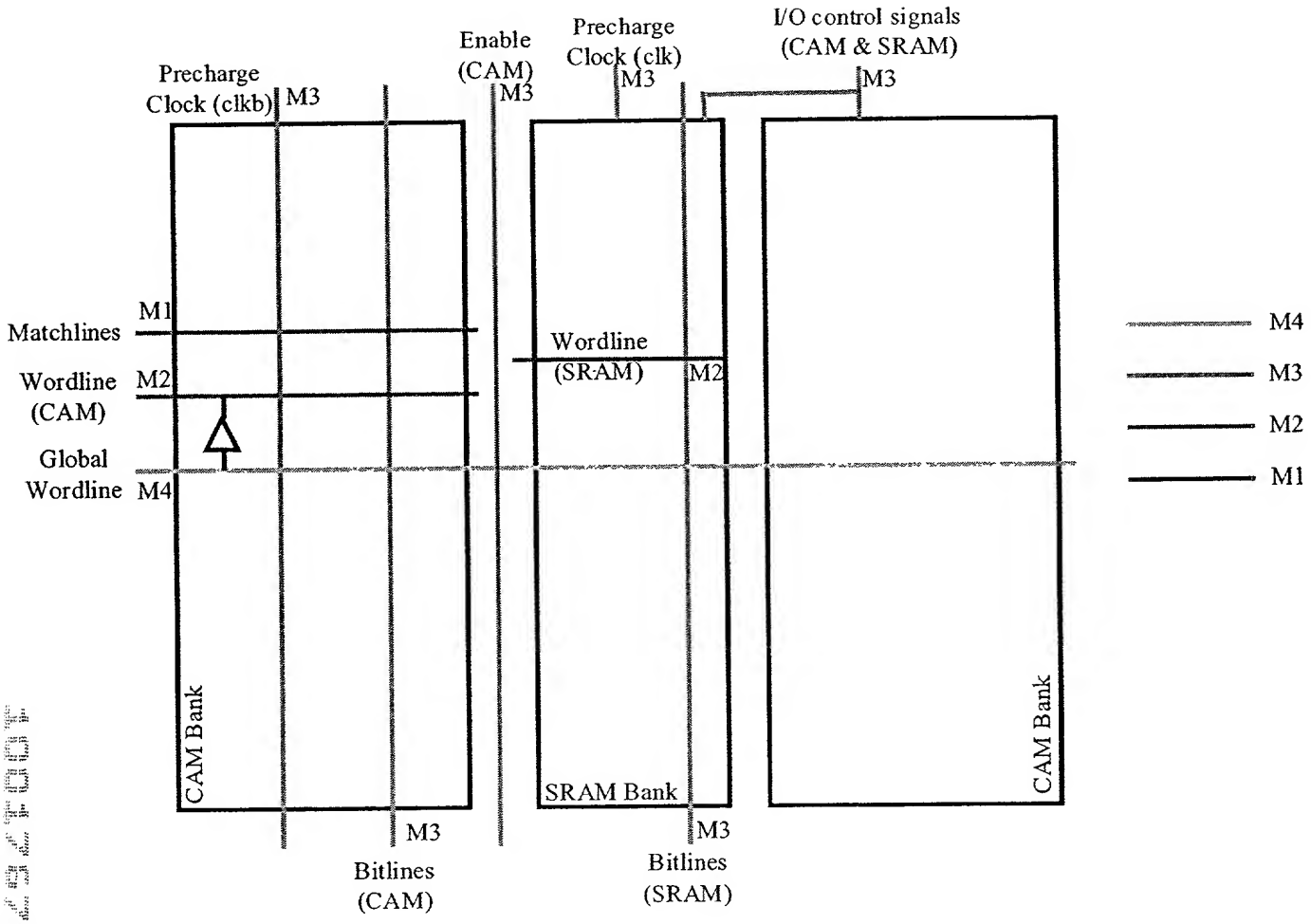


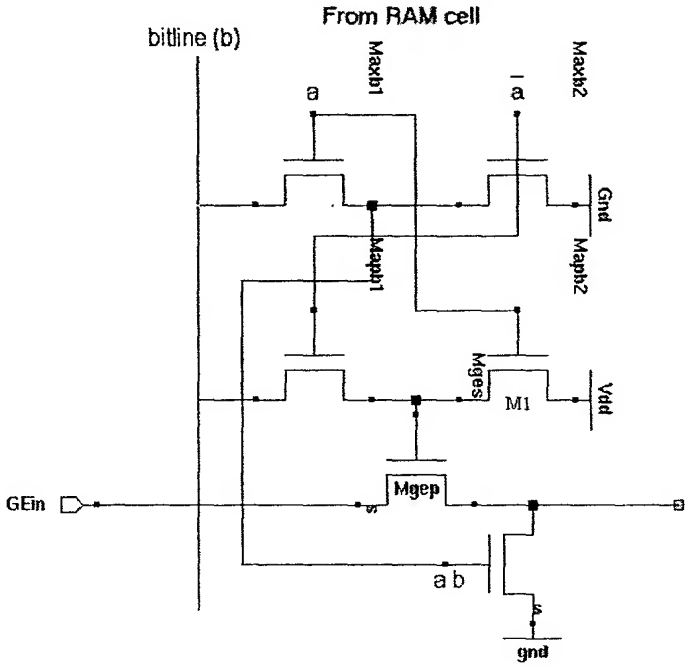
Figure 4 : Placement of the wires for the unit

logic terms, this means that we have to 2's complement one of our numbers and add them together. If the overall addition result is positive (i.e. the extra bit for 2's complement is 1) there would be a carry generated, otherwise there would be no carry. We used a carry-chain architecture to implement our CCM.

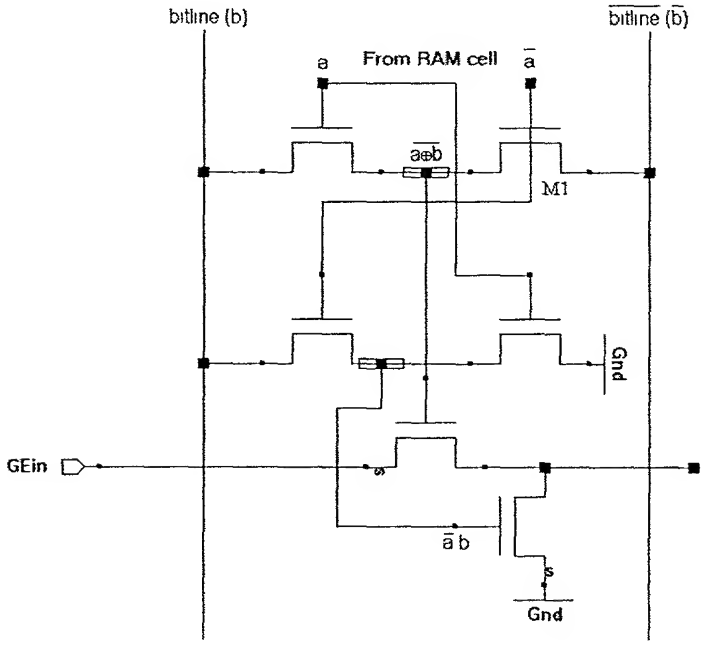
It is not desirable to do a 2's complement operation on the IP number for each lookup. One solution is doing the 2's complement operation on the CCM content when it is stored during an update. Another solution is storing the original CCM content, but do the carry chain logic operations on the inverse of the stored value. In this case there should be a carry input to the carry chain. Since 2's complement of a binary number is equal to bitwise inverse of that number plus 1, the end result will be the same as the first solution.

Effectively the CCM content is subtracted from the IP number each time and a carry is generated if the IP number is Greater than or Equal to CCM content, hence the name G.E.CAM. Two possible implementations are shown in Fig. 5. Figs. (a) and (b) correspond to first and

second solutions respectively. In both cases transistor M1 can be connected either to Vdd (done in Fig. 5(a) implementation) or to $\overline{bitline}$ (Fig. 5(b)). Connection to $\overline{bitline}$ may make the overall cell size smaller. Of course there could be other implementations for generating the inputs to the series and parallel carry chain transistors.



(a)



(b)

Figure 5 – CCM carry chain

We used one phase clocking scheme. Each section has one whole clock cycle to perform its operation. Static logic sections (like the decoder and Priority Encoder) have the whole cycle time to evaluate and so their timing is not critical. For precharge logic sections (like the CCM and CAM), the clock low cycle is used for precharge and clock high is used for evaluation.

To save power our bitlines are not precharged (this will cut down the power by half). With this scheme it seems that there is no guarantee that all the nodes in the matchline chain are precharged. To eliminate the potential charge sharing problem, we send the bitline data during the precharge period so that the matchline nodes which are going to be connected to the matchline outputs are precharged correctly.

As was discussed in the previous section and also from the above paragraph, we have two close critical paths, the CCM operation and the CAM bank evaluation. Both were simulated in spice. For CCM, the whole path was simulated, from the output of the previous stage flip flop to the input of the next stage flip flop. Since it was observed that the number of addresses with more than 24 bit prefix lengths are very small (0.1% from statistics), we decided, without loss of generality, to allocate a few bins for more than 24 bit prefix lengths and then use 24 bit CAM's and CCM's for storing the rest of the IP numbers. Since the word length for all CCM's in this implementatin was 24 bit, we did the simulation for this word length.

For CAM, simulation path was from the output of the previous stage to the input of sense amplifier (one row of data). The delay of the sense amp was then estimated and added to the overall delay number. Since the longest word length of CAM cells is 32 bit, we did our simulation for 32 bit long word.

For inputs we assumed that they all have 400ps rise and fall time (FO4 rise time in TTSS corner in which we are measuring the speed of the circuitry). For output of each section we used appropriate loading.

All the wire capacitances and resistances were included in the simulation. These were extracted from the general wiring scheme discussed in previous sections. For capacitances worst value was considered, that is it was assumed that the metal layer is sandwiched between top and bottom metal layers. For all cases the worst case delay was simulated, i.e. it was assumed that the

signal should travel the whole length of wire, although for some loads it had to travel shorter distances.

Since we simulated one complete row in CAM, for many of the logic circuitry like the SRAM read-write circuitry the correct load and fanout are already there in the circuit. For signals and controls that go to all the rows (like bitlines, enable signals and clock signals), the necessary fanout was simulated by using dummy gates and capacitive loads.

For measuring the critical path delay, all the initial conditions were set to the opposite of the final values during the cycle we are simulating. This way all the precharge and evaluation times are taken into account correctly.

We also did the simulation for the write operation in the CAM banks (including the decoding) and Priority Encoder, although from initial hand calculations it was obvious that the timing of these circuits is not an issue.

5.1 Circuit Design, and Critical Path Simulation

5.1.1 CCM Simulation

The circuit schematic for each cell is shown in Fig. 6. Since there is no read operation from the CCM cell, there is no need to have more than minimum size NMOS's in the SRAM cell. For the logical operations (bitline.D) and (bitline+D) transmission gate logics were used to save area. The GE line (equivalent of matchline in normal CAM) transistors were chosen to be 16λ in size in order to decrease the delay. The GE lines are implemented with minimum width (for lower capacitance) M1, since the distances are not that far. The wire capacitance and resistance were also included in the unit cell. Wire capacitance was assumed to be Cgnd + 2*Cadj, because during the evaluation half cycle all the nearby lines are silent.

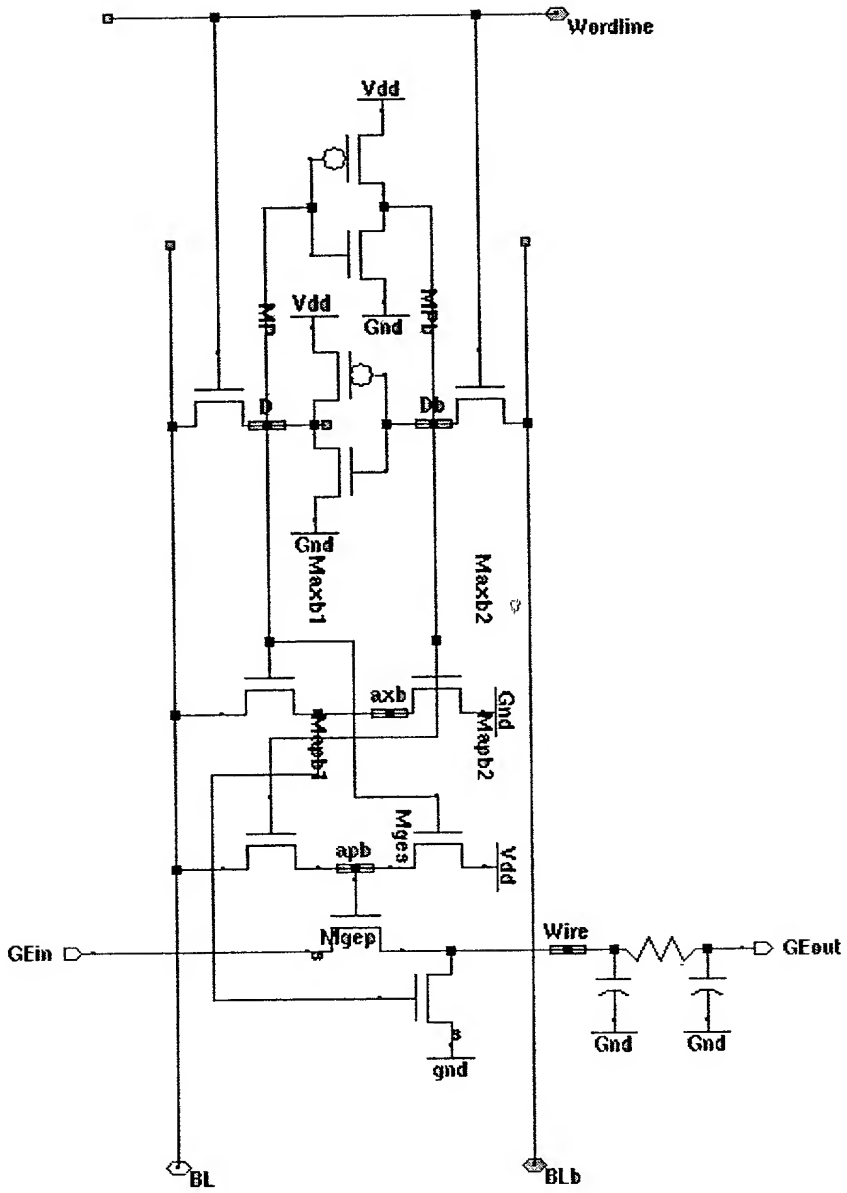


Figure 6 – CCM unit cell

The bitline circuitry is shown in Fig. 7. The bitline inverter was chosen to be minimum size because the capacitance on this bitline is only equivalent to 62λ on bitline and 14λ on bitline_bar. Eight of these cells are connected to each other to form a CCM 8 bit block. The wordline wire load for this 8 bit section is shown in Fig. 8.

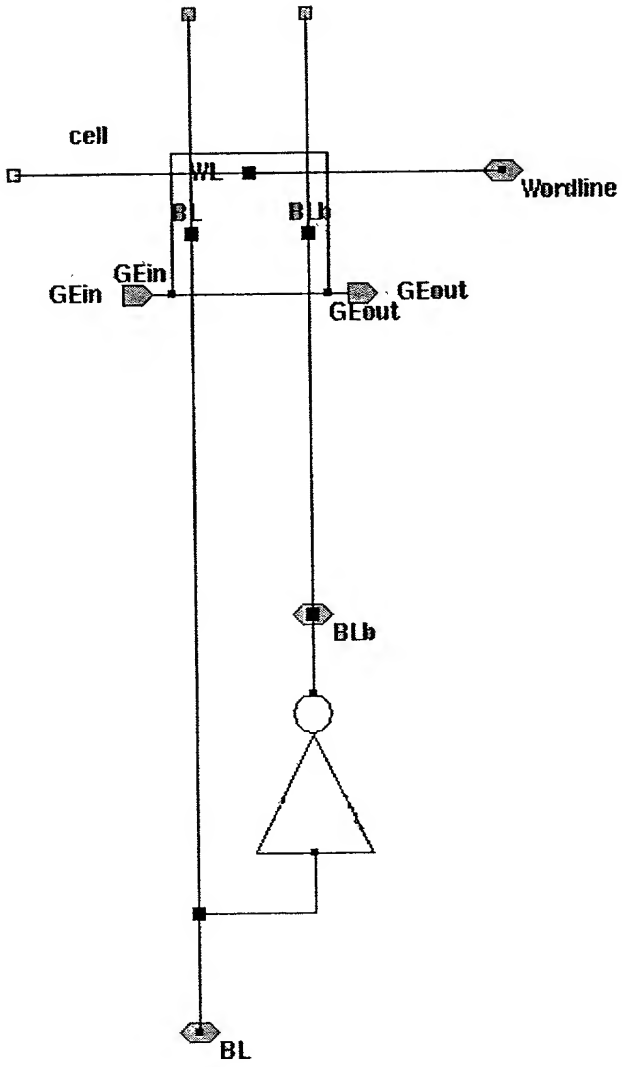


Fig. 7 – The CCM cell with bitline circuitry

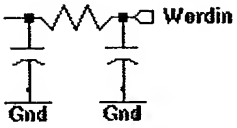


Fig. 8 – The wordline wire load for 8 bit section of CCM

Connecting all these 24 bits (3 8-bit sections) will result in a large delay (around 10 ns). So matchline buffering is needed to reduce this time. Buffers are put between each 8-bit section, with the total of 2 buffer sets (2 inverters for each set). The buffers between 2 sections are shown in Fig. 9.

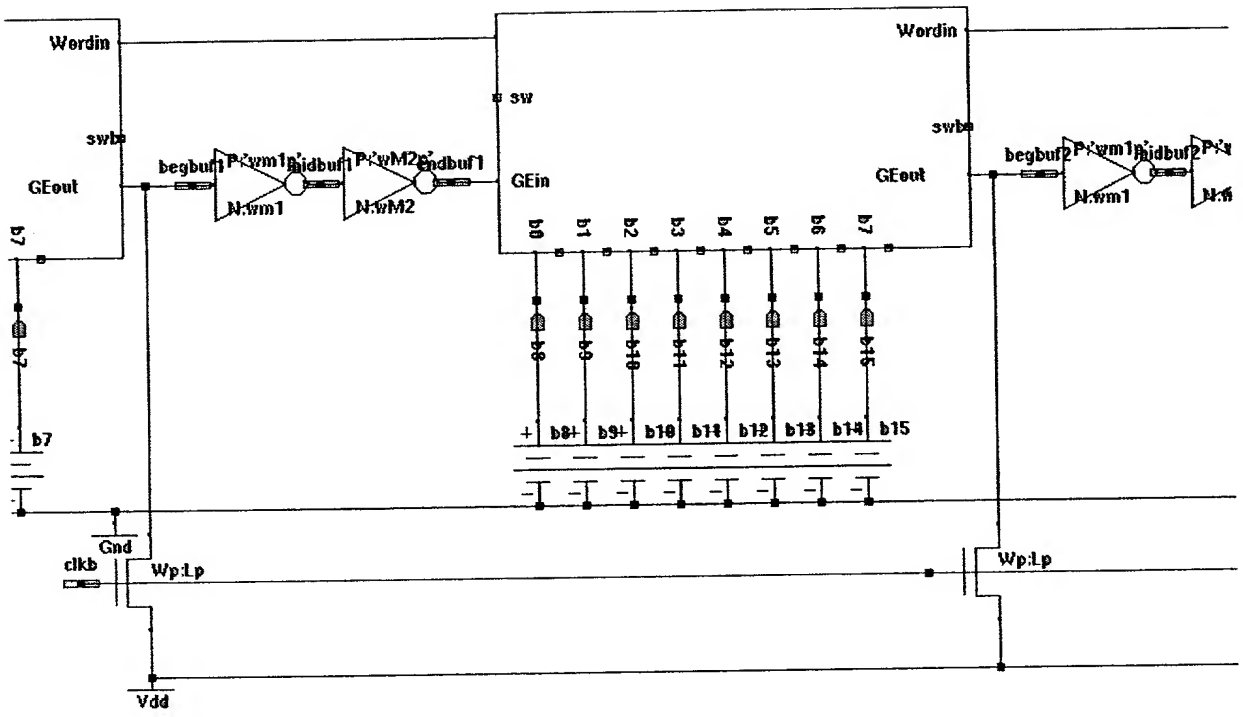


Figure 9 – 8-bit blocks with buffers between them

Now the question is that should we precharge our GE lines low and then charge them high for a GE (Greater than or Equal to) or should we do it other way around, precharge them high and then discharge them when a GE occurs? If we had no intermediate buffers (only a final buffer), then maybe precharging GE lines low and then charging them high makes sense. That way the lines initially go up very fast (because transistors are on) but reach the final value very late.

But this option is not good if we have buffers in between. In a test spice simulation, even though Domino gates were used for buffering, the delay did not improve from the no buffer case by that much. The problem is that even if the buffers are skewed in such a way that they switch at a low threshold value, their input signal will never be very strong and so the buffer stages have a long delay. So because of these concerns the lines were precharged high and then discharged in case of a GE. The precharge transistor is also shown in Fig. 9. The precharge is done through an NMOS transistor. The reason is that we do not want to charge our GE line more than Vdd-Vt. Charging bit lines more than that will turn off the GE line transistors even more and will slow down the GE line.

Each buffer set consists of two inverters. At first glance it seems that the first inverter should be skewed such that it switches at a high threshold voltage. But the fact is that the GE line never charges up more than Vdd-2Vt which can be very low (Around 2volts for Vdd=3volts). So the first inverter should be actually skewed such that the threshold level is lower than normal inverter to provide enough noise margin. By doing a set of simulation sweeps, the W_{PMOS}/W_{NMOS} was chosen to be 0.2 to provide a switching point of around 1 volt (a noise margin of about 1 volt). The second inverter was skewed the same way but this time for speeding up the circuit. The sizes where chosen to be $W_{NMOS}=20\lambda$ and $W_{PMOS}=4\lambda$ for both inverters. For the inverter at the end of GE line the same sizes were used. These were obtained by optimizing the RC line delay model.

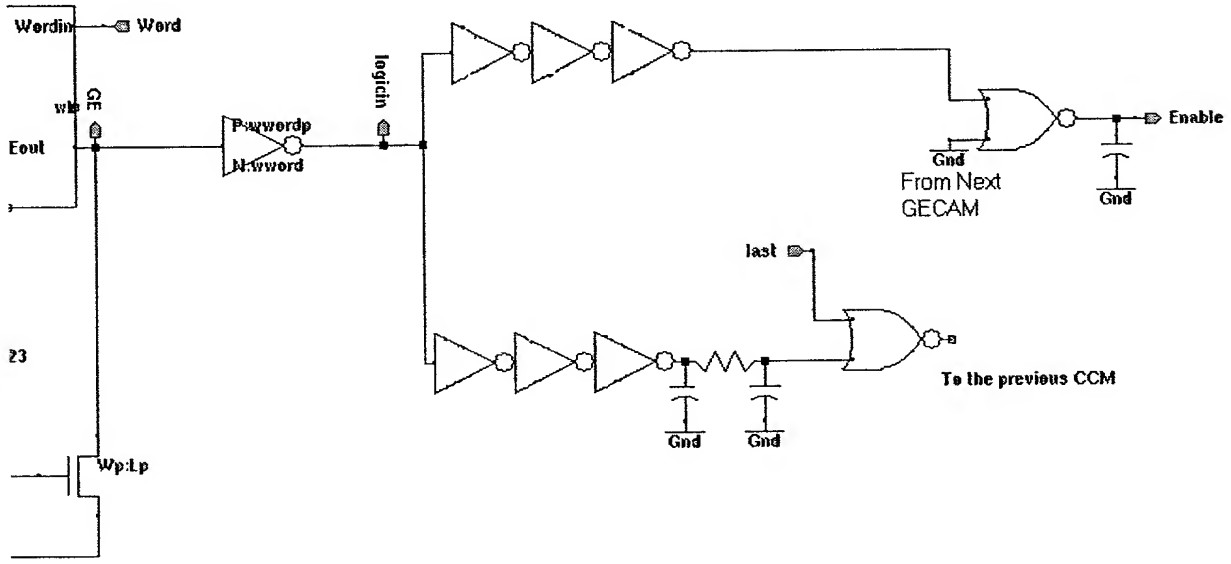


Figure 10 – The output logic for CCM

The output logic for CCM is shown in figure 10. The output of this CCM should be sent to the logic of the previous CAM bank's CCM. The wire model is shown in the figure. The output is buffered before sending it to the wire to minimize delay. The buffers are also sized for best delay. The buffers in the other path are added intentionally to equalize the delay of both paths as much as possible. If this is not done, then there is a possibility of creating glitches in the enable output (when GE line is discharged for both this CCM and the next one, but the next one

arrives later because of wire delay). These unwanted outputs actually slow down the circuit. With the sizes given, there was no unnecessary transition on Enable output. The margin was so much that even if one of the paths become slightly faster, the Enable glitch will be very small.

As we see in Fig. 10 there is a 'last' signal in the logic. If the 'last' signal is 1 in the next CAM bank, then the enable output will only depend on the GE from current CCM. This is explained in the architecture section. This 'last' signal is stored in MSB of mask bit, since the 8 MSB's are not used in masking (there are no prefix lengths less than 8).

The clock circuitry is shown in Fig. 11. The wire loading is also included in the model (for the wire which should go through the height of the CCM cell).

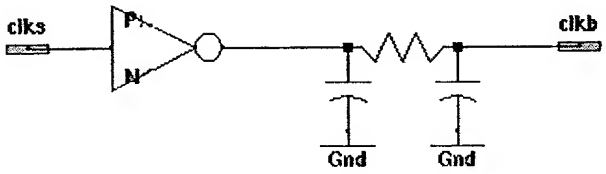


Figure 11 – Clock circuitry for CCM

The delay was measured for the worst case GE operation, i.e. the carry is generated in the LSB and should propagate all the way to the end. The measured delay from input to enable output was measured to be 2.64ns. This was measured in TTSS corner.

Since the threshold voltage of the first inverter in each buffer set was set to be low (by weakening the PMOS) we checked the SFSS corner. The circuit worked properly.

The input capacitance for clock is equivalent to 18λ. The input capacitance for bitline is 74λ.

5.1.2 Normal CAM Simulation

Many issues in CAM design are similar to the CCM's issues and so are already discussed. The basic CAM cell is shown in Fig. 12. As explained before we have both 32 bit and 24 bit CAM banks. Since the 32 bit one is more critical, we will simulate this one.

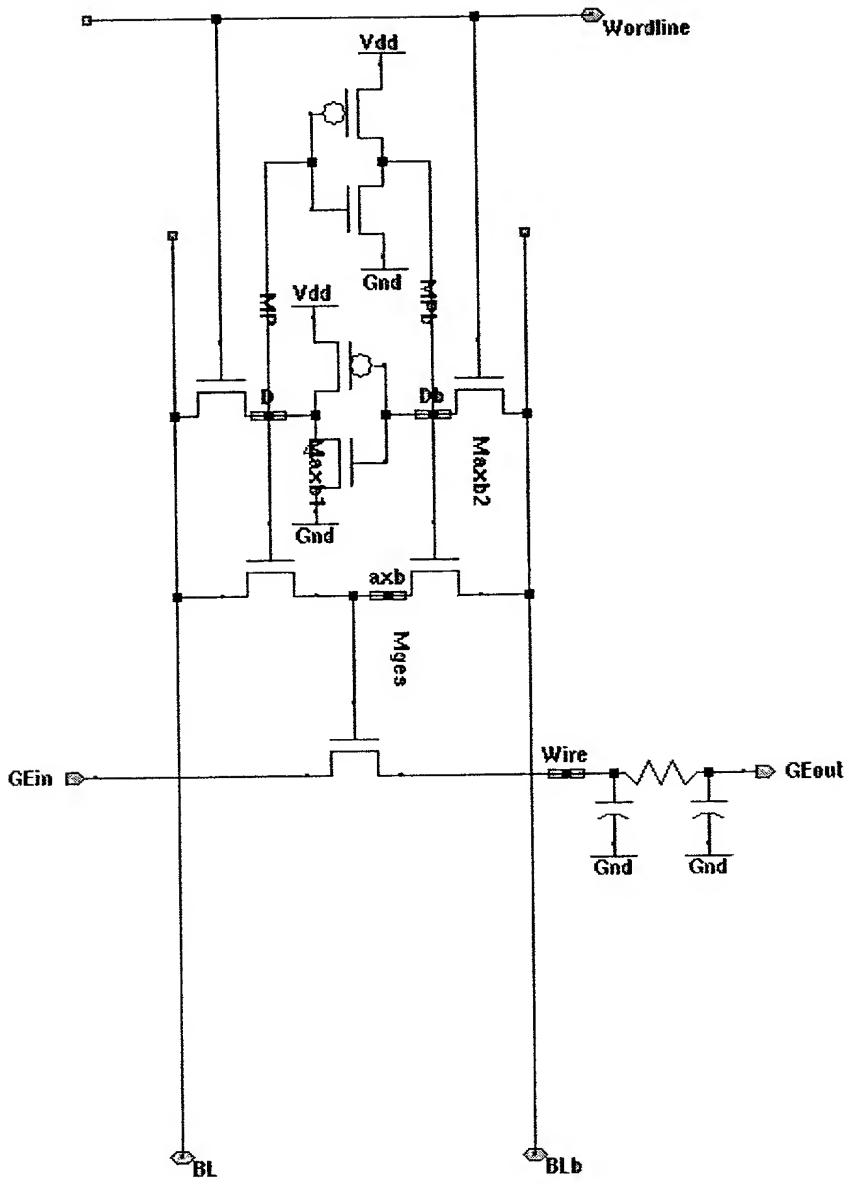


Fig. 12 – Basic CAM cell

The matchline wire model is the same as the GE line in CCM. Choosing the NMOS transistor size as minimum size (4λ) helps minimizing the area, specially that CAM cell is covering most of the chip area. The matchline transistor and the transmission gates are also chosen to be 8λ. There are made as small as possible to meet the 7ns cycle time objective. The loading of other 99 rows on the bitline and also the bitline wire capacitance and resistance are

modeled as shown in Fig. 13. For wire capacitance still the formula Ctotal=Cgnd + 2*Cadj was used (worst case assuming M2 below and M4 above). Since the two bitlines are switching in opposite directions, it seems that Ctotal = Cgnd + 4*Cadj should be used for wire capacitance. But since these bitlines are around 25λ away from each other, Cadj is pretty small and can be ignored.

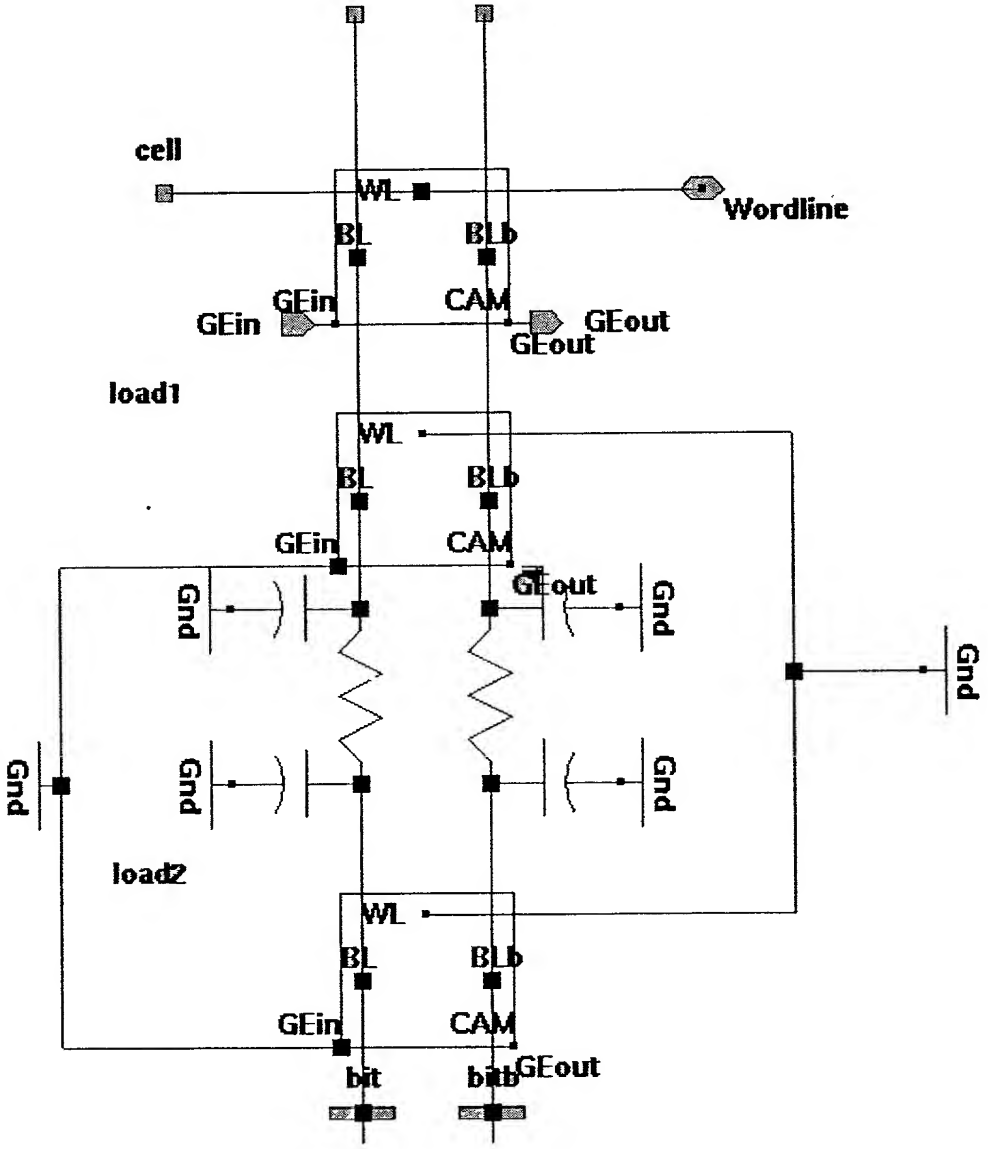


Fig. 13 – Modeling the loading on the bitlines

The bitline input circuitry is shown in Fig. 14. The mask implementation is also shown in this figure. Only 24 bits of CAM have this implementation and for the other 8 (MSB), mask bit is always 1. The mask bits are stored in an extra row of SRAM cells. When 'mask' bit is 0, then

both bitlines will be charged to high. This way the input of the matchline transistor will be always high as desired for the masked bits.

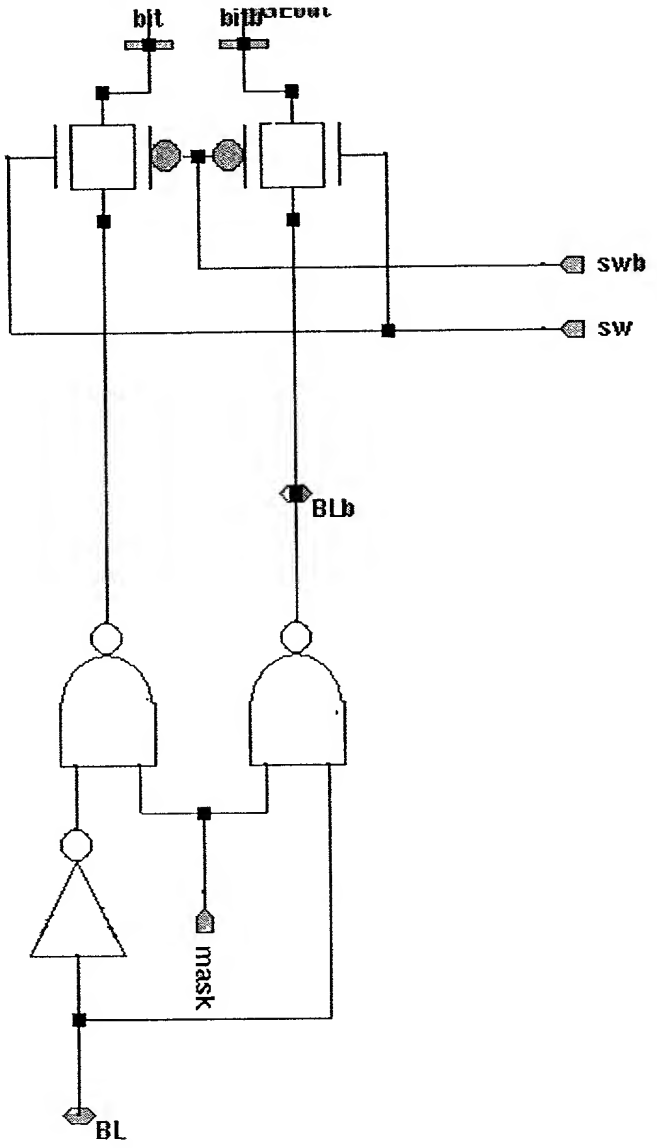


Figure 14 – CAM Bitline input circuitry

For designing the drivers we did not want to minimize the delay of the drivers, because they are operating during the precharge time and so their timing is not critical. Our only constraint were having an input capacitance of 200λ (the acceptable output capacitance for the FF we designed in HW #4) and also performing the required logic. We also tried to equalize the

delay of the two paths. The delay of the bitline path is almost 7.4FO4 and the bitline_bar is around 7.8FO4 (1.4ns). This delay time is acceptable considering our cycle time.

For driving the CAM bitline switches, the circuit shown in Fig. 15. is used. When Update signal is high it means that we want to write into the CAM and so the switches are turned on. Likewise when Enable signal is high it means we want to do a lookup operation and so again the bitline switch is turned on. When Enable signal is low we do not write anything to the bitline to save power. The output of this logic sees the capacitance of a wire the length of the CAM bank width. The delay of the 2 paths are equalized. The 'Swcam' delay is around 3.7 FO4 and the 'Swbcam' delay is around 3.9 FO4 delays. The input capacitances were assumed to be 100λ for both 'Update' and 'Enable' signals to be within the range of 200λ load for FF.

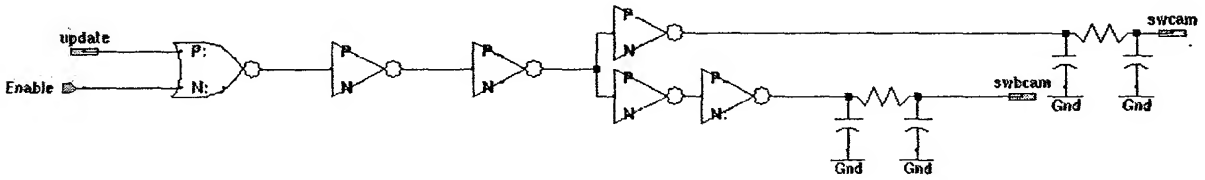


Figure 15 – CAM bitline switches circuitry

For speeding up the CAM operation, we divided the 32 bit CAM to 2 16bit CAM blocks and then NORed the output results. Since our SRAM cells are shorter in height than the CAM cells, there is enough space below each SRAM row for an extra pitch of wire. So as shown before, the 2 16 bit CAM banks are placed on the two sides of the SRAM bank and the output of one of them is passed below the SRAM cells to the other side. The output logic of the CAM blocks is shown in Fig. 16. Each 16 bit block is also divided to 2 8bit blocks and buffers are used between these 2 blocks, the same way as CCM design.

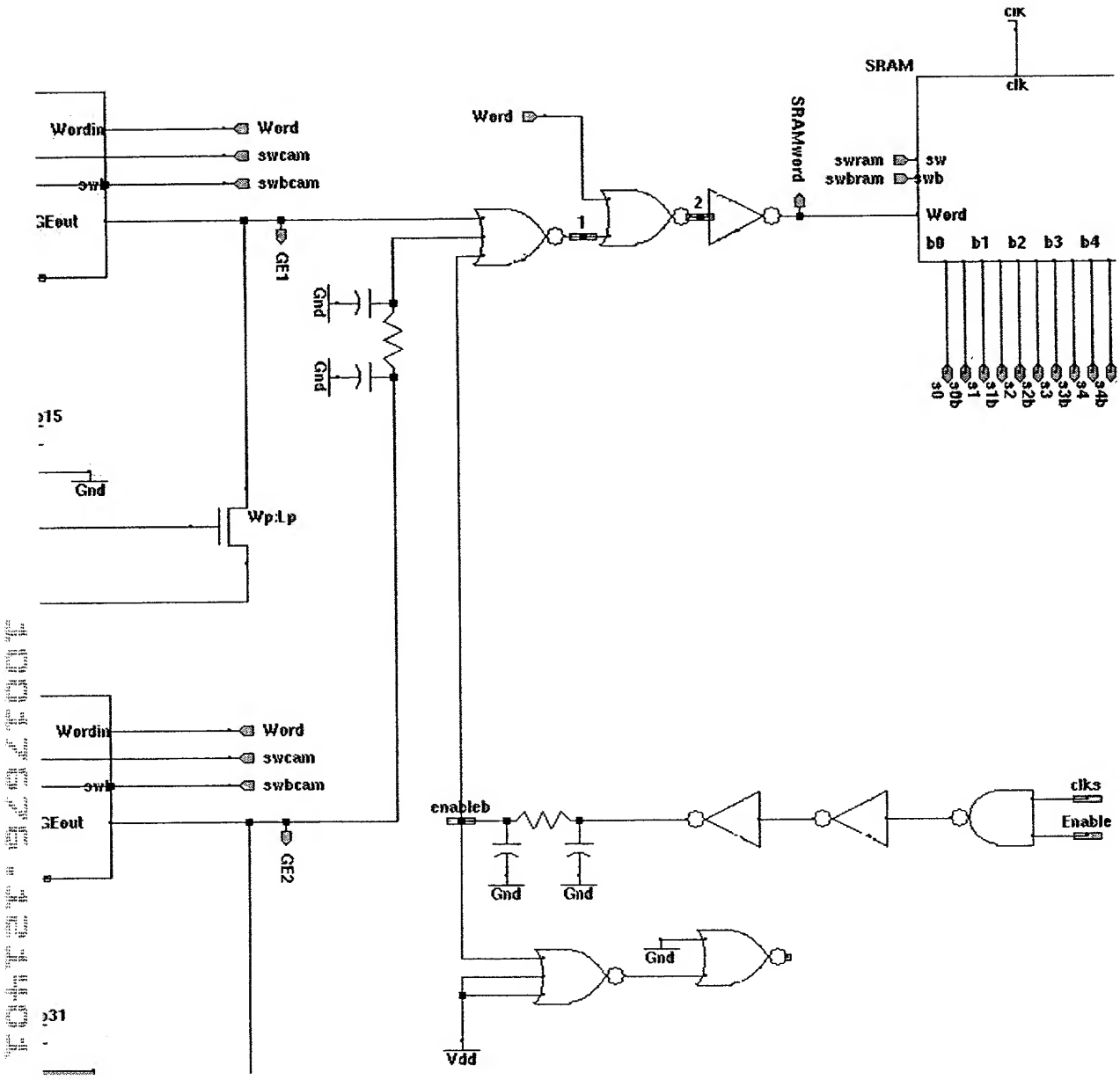


Figure 16 – CAM output logic

Since we do not want to activate SRAM wordlines when the CAM bank is not active, we included the 'enable_bar' signal in the NOR operation. This way when 'enable'=high, the circuit operates and when 'enable'=low, the SRAM wordline remains low. Also the wordline of the CAM is NORed in the second NOR gate for the write operation in the SRAM during an update phase.

For the first NOR, the sizes were chosen so that the threshold voltage is low enough to avoid the problem discussed in previous section. The apparent effective PMOS to NMOS ratio here is 12/(4/3)=9 which is more than 5 which was assumed for inverter buffer gates. The reason is that when for example GE1 is still high (around 2 volts), 'enable_bar' and GE2 can be both low, which turns on the 2 PMOS transistors and effectively decreases the PMOS chain resistance.

Since the logical effort of the first NOR is very high (around 8), the next NOR gate is chosen to be minimum size. The overall LE from input of second NOR to the wordline is around 55/20*5/3=4.6. The gates are sized to get minimum delay. The overall delay of the logic is estimated to be around 5FO4, of which more than 3FO4 is from the first NAND.

In Fig. 16 logic for generating 'enable_bar' is also shown. At the input, the clk signal is NANDed with 'enable' signal so that when clk is low (precharge state), the 'enable_bar' signal is kept high and so SRAM wordline is not activated erroneously. The delay of this chain is not critical as long as it is shorter than the delay of the matchline. The worst case wire loading (the full height of the Cam bank) and also the fanout of all the skewed NOR gates are also modeled. This delay is around 3FO4 which is less than the delay of GE lines and so it is ok.

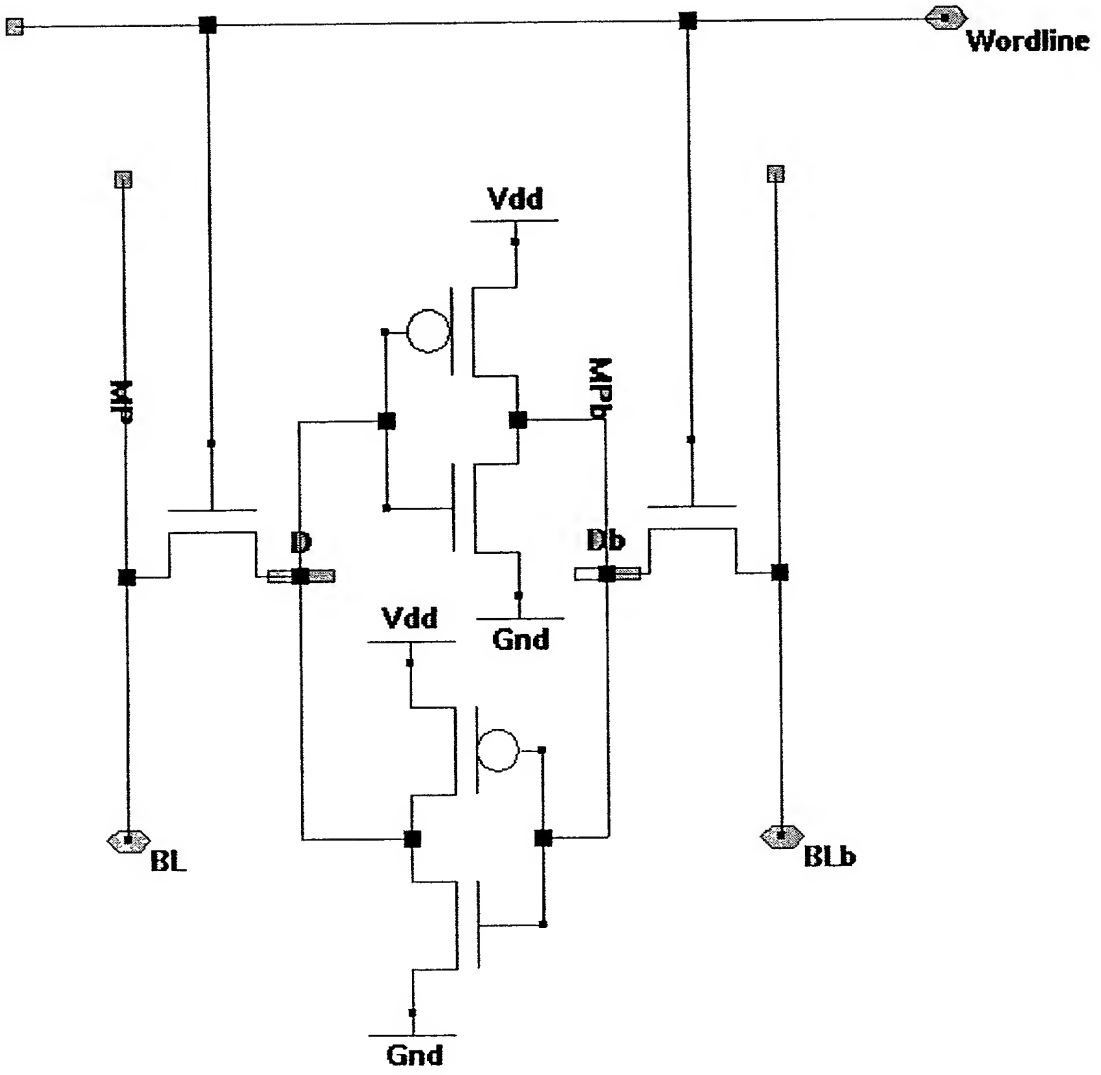
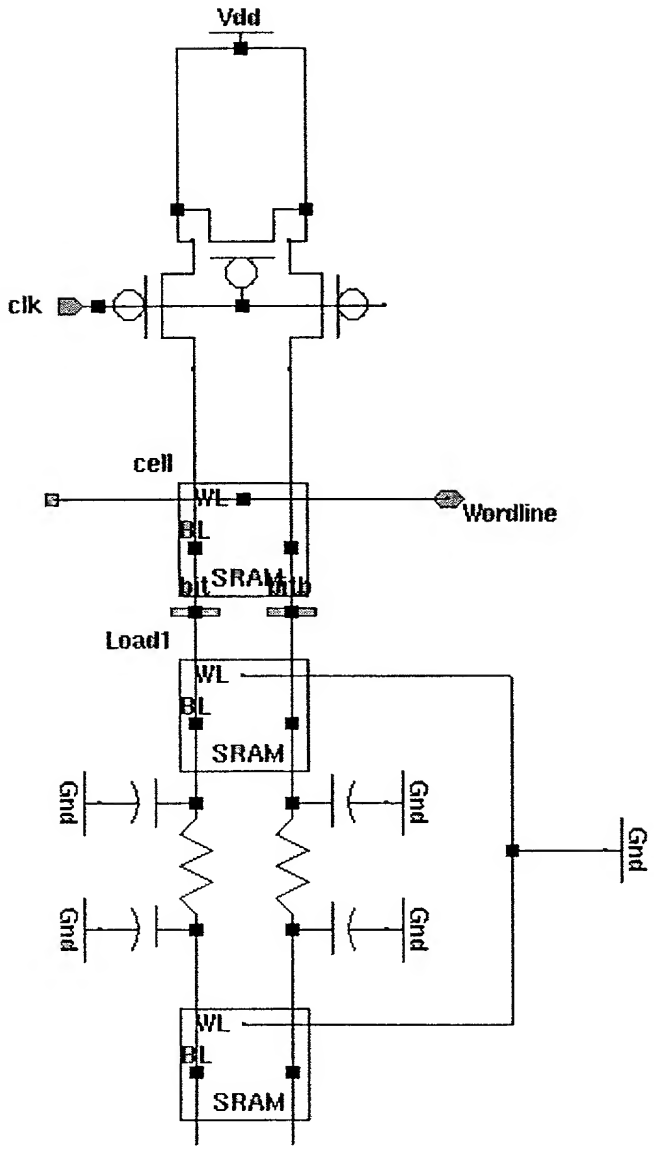


Figure 17 – Basic SRAM cell

The basic SRAM cell is shown in Fig. 17. The NMOS size of the cell (8λ) was chosen such that during a read operation, the low voltage on the SRAM output does not increase more than 500mV. This gives enough margin such that even if the pass gate transistor becomes stronger due to mismatch and process variations, still the read operation is done without destroying the SRAM content.

The modeling of the bitline wire load and also the loading of the other rows is done the same way as the CAM cell and is shown in Fig. 18. Precharge transistors are added for read operation.



Figre 18 – SRAM with wire and load from other rows

The SRAM read and write circuitry is shown in Fig. 19.

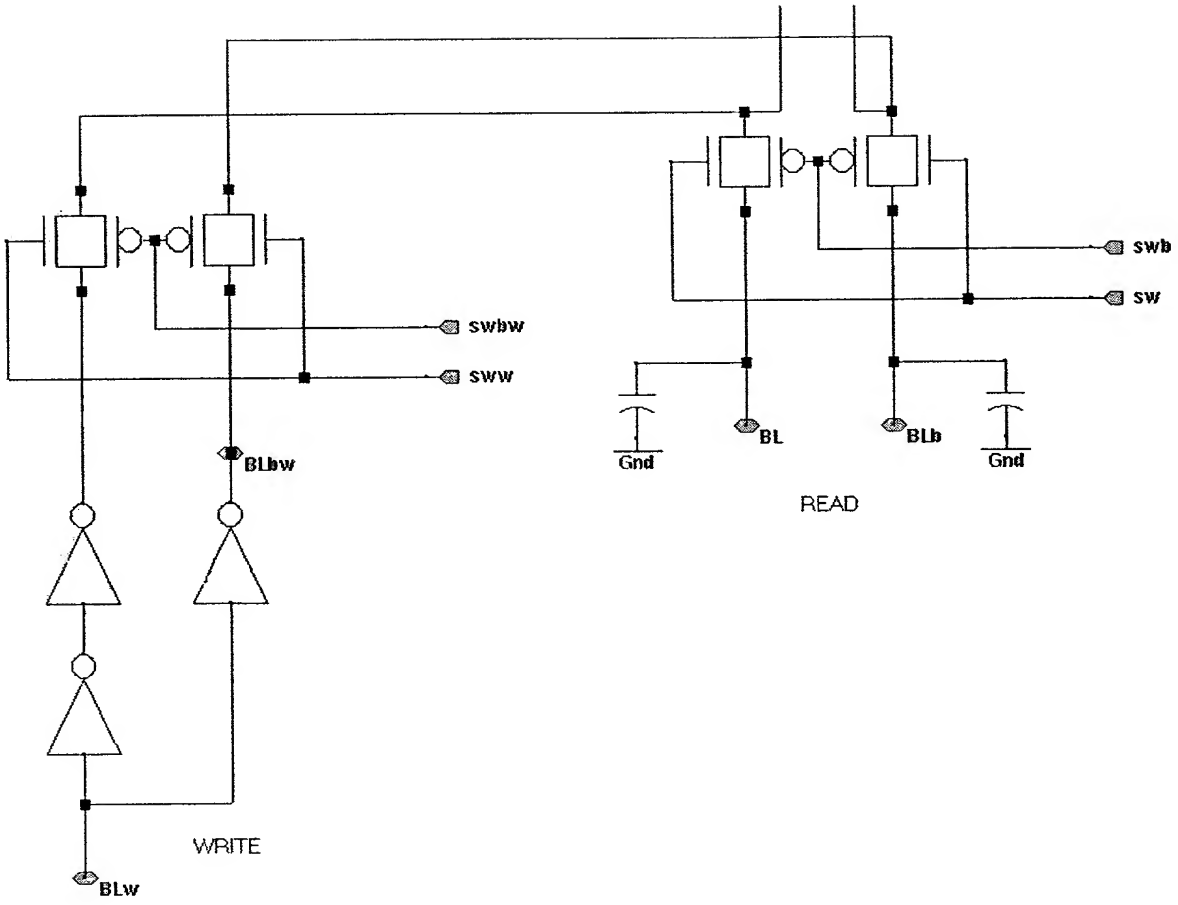


Figure 19 – SRAM read and write circuitry

For read circuitry, we have 2 switches which open the bitlines to the input of the Sense Amplifier (not shown). The 30fF capacitors are modeling the sense amp input capacitors.

For designing the write drivers, again like the CAM write circuitry, we did not have to minimize the delay. Since the write circuitry is static, it has one whole cycle to complete its operation. Our only constraint was bitline input capacitance (60λ in this case). The delay of the fork is equalized. The delay of bitline is around 3.7 FO4 and the bitline_bar around 3.2 FO4 from hand calculations.

The SRAM read and write switches should be driven. The driving circuits are shown in Fig. 20. One is driven by 'Enable' to during a read operation and the other one is driven by 'Update' for a write operation. The outputs of these logics see the capacitance of a wire the length of the SRAM bank width (plus the switch transistors input capacitances). The delay of the

2 paths are equalized. The 'Swram' delay is around 2.9 FO4 and the 'Swbram' delay is around 2.7 FO4 delays. For 'Swramw' and 'Swbramw' this delay is almost 1.9 FO4.

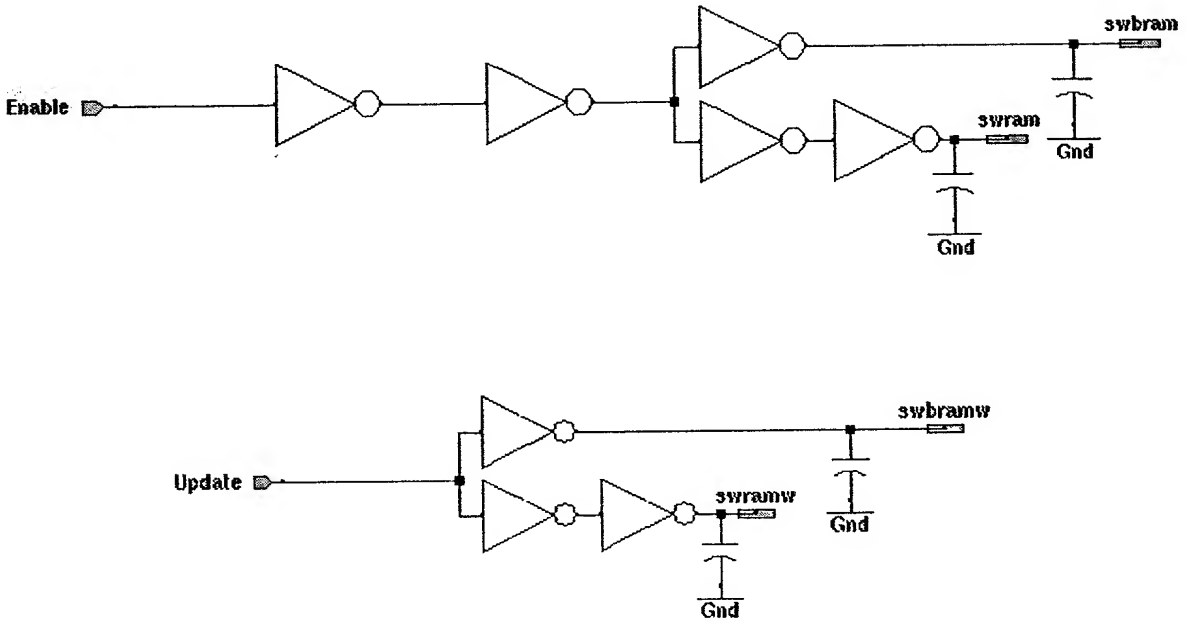


Figure 20 – SRAM read and write switches driving circuitry

To further save power during the period which the CAM bank is inactive (not enabled), the clock is also shut off when 'enable' is low. The circuit shown in Fig. 21 is used for both driving the clock load and shutting off the clock. When 'enable' is low, 'clk' is kept high and 'clkb' is kept low and so there will not be any precharge.

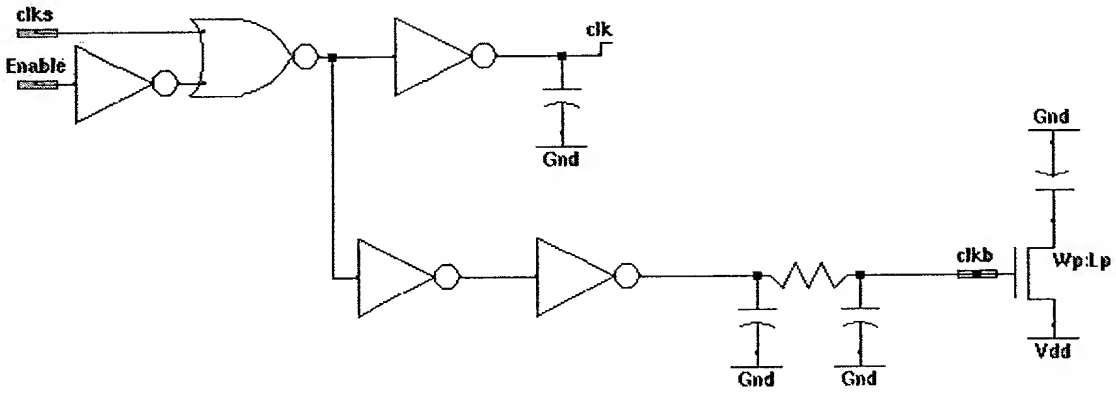


Figure 21 – Clock drive circuitry

The outputs of 'clk' sees the capacitance of a wire the length of the SRAM bank width and the SRAM bitline precharge transistors. The delay for 'clk' signal is around 4 FO4.

The 'clkb' signal output sees the load from a wire which runs the full height of the CAM bank. It also sees 400 precharge transistors, out of which 4 transistors are present in the circuit. The other 396 are modeled by a dummy transistor load. The capacitor load of this dummy transistor is chosen to be very small because in actual case also only one of the matchlines should be precharged and the others should be at their high value. The delay for 'clkb' signal is around 4 FO4.

The delay of the lookup was measured by one of these 2 methods: 1- The delay from the rising edge of the clock to the rising of SRAM wordline was measured and then 0.65ns was added for the Sense Amp read operation , or 2- The delay from the rising edge of the clock to the point where the SRAM bitline output changes by 250mV was measured. These numbers (0.65ns delay or 250mV were both extracted from EE313 notes). Both methods resulted in the same delay value. The overall evaluation delay of the lookup operation is around 3.5 ns from the clock edge (in TTSS corner). BUT this delay from the falling edge of 'clkb' signal is only 2.65ns. So if the FF for this stage is driven by this delayed clocked we can borrow some time from the next clock cycle, as shown in Fig. 22. Since the next operation is Priority Encoder and that operation is fully static, it can afford to lend some time to the previous operation.

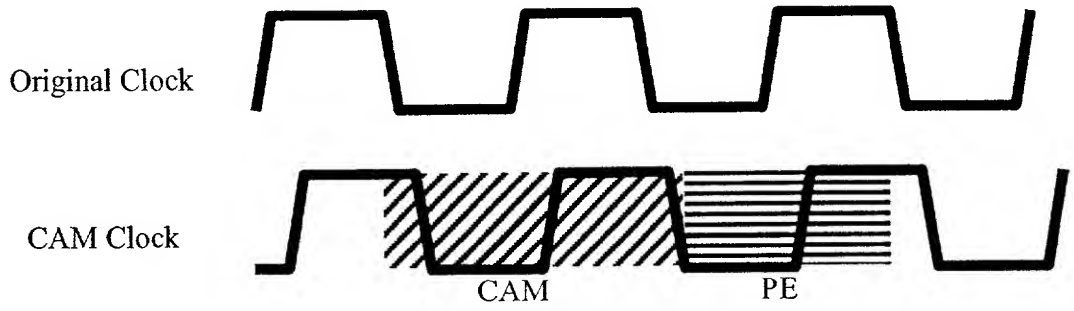


Figure 22 – Cycle Borrowing

The 2.65ns operation allows us to operate at a 7ns cycle time (considering 1.5 FO4 clock skew, 2FO4 FF delay and 45% cycle time).

The input capacitances for this stage are as follows: Clock=196λ≈78fF, Enable=250λ≈100fF, Update=235λ≈94fF.

5.1.3 Decoder design and Memory write timing

The decoding is done in 2 cycles. First 8 bit decoding is done to choose one of the 256 banks on the chip (the structure is shown earlier in the report). In the next cycle, a 7 bit decoding is done to choose the row of CAM-SRAM bank. In the same cycle the data is written in the CAM and SRAM. So this second decoding is more critical than the first one. The 7 bit address space is divided to a 3bit and 4bit address spaces. So we predecode on 3bit, then predecode on 4bit and then combine them. The 3 bit predecode is faster than the 4 bit one, so the circuit for the 4-bit predecode is designed. The circuit is shown in Fig. 23.

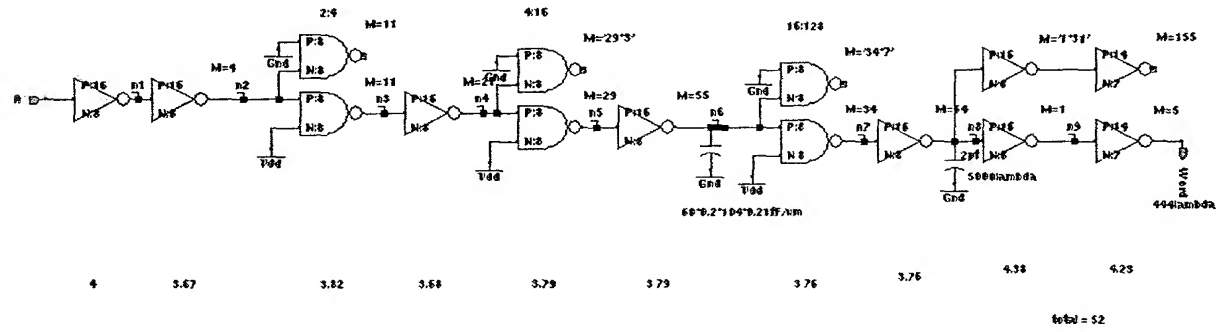


Figure 23 – 4:16 Decoder

First a 2:4 decoding is done. Then a 4:16 decoding is performed. The 16 resulting lines are then run through the whole height of the CAM –SRAM bank to do the 16.8=128 decoding. The equivalent wire capacitance is 262fF which is used in the simulation. After the final decoding stage, the global wordline is driven. The capacitance of this line (almost 1cm long) is around 2pF. The resistance of this line is around 125 ohms, which was ignored. Considering this resistance in our sizing calculations causes the buffer stage driving global wordline to become very large, which is unnecessary, since the decoding time is not that critical. The wire resistance adds around 2-3FO4 delay to the total delay of 10 FO4, which is well within acceptable range. The simulated decoding time is 2ns and the overall write time is measured as 2.53ns. So writing

operation is not a problem and can be done well within the intended cycle time of 7ns. (Remember that both the decoding and write operations are static operations and so can be done during the whole cycle).

Since the write operation depends on the strength of the cell PMOS and the pass gate NMOS (NMOS is stronger), in SFSS corner write operation may fail. This corner was checked and the write operation was done correctly.

The address input capacitance is 24λ (9.6fF).

5.1.4 Priority Encoder

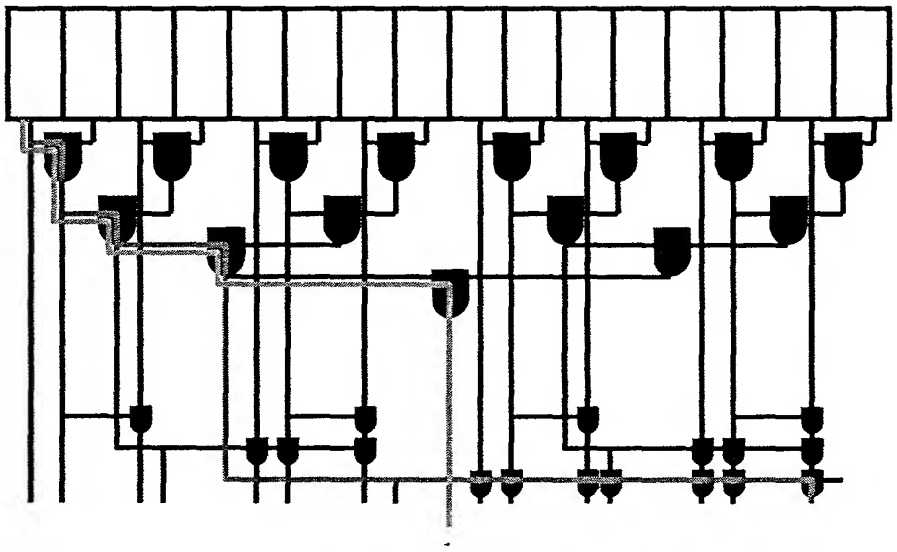


Figure 24 – Priority Encoder Tree Structure

Our priority encoder encodes priority among the CAM banks. Since these are distributed around the chip there are likely to be very long wires somewhere in the decoder. Following the principle of putting the wire capacitances as far in the gate chain as possible, all the gates are nearly always dominated by wire capacitances (expect for certain gates with huge fanouts). Also since we have only one priority encoder, area is not really an issue. So, we went in for a distributed static tree PE. This also allowed this pipeline stage to lend some time to the CAM search stage increasing our lookup speed further. For simplicity of design as well as power considerations we went in for a two stage 16x16 (256) decoders as shown in the floorplan. Between the two decoders we had 10 AND gate stages so the available time was divide equally between these

gates. Taking 7ns cycle time, giving 0.8ns to the CAM stage and accounting for flipflop delays, clock skew and uneven clock duty cycle we have to meet a total delay spec of 5.4ns. While we wanted to be comfortably inside this spec without wasting area and power on large fast gates so we choose 3.5ns as the target.

A typical stage (AND gate) in the PE looks as shown in figure 25.

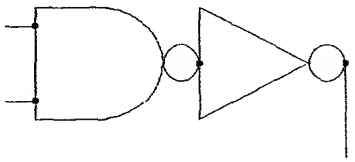


Figure 25 - Typical stage in the PE (AND gate)

To get a total delay of 350ps ($\sim 10\tau_{inv}$) per stage writing the delay equations we get a fanout of 9.1 across the whole gate. The wire capacitances and gate load were calculated from figure 24 by tracing the positioning of each gate relative to the layout on the chip. The critical paths are shown in the figure 24. The green path is the critical path for the first stage and the red is that for the second stage. Simulations at the TTSS corner gave a delay of 3.8ns including all the wire capacitances meeting the spec comfortably.

5.2 Power Estimate

The main sources of power consumption are as follows:

1- Bitlines of CCM 's and CAM (and SRAM's) and other lines inside them

2- Global wires inside the chip, mainly the IP number bus, in this category there are also global word lines but they are only active during the update. Since update frequency is a very small fraction of lookup frequency, we can ignore this power.

3- Power consumed in the logic, like in Priority Encoder and CAM logic. The same as above, we can ignore the power burnt in the decoder.

We simulated the CAM's, CCM and Priority Encoders at the TTTT corner and integrated the current through Vdd to obtain the power. Here are the results:

1- For 32 bit 100 entry CAM bank, we obtained the average energy per cycle to be 0.15 nJ and the average energy per cycle when all the bitlines switch to be 0.83 nJ. So 0.68nJ is due to bitlines. Since at any time half the bitlines switch on average, then bitlines consume about 0.34nJ per cycle. So the average consumption per cycle for 32 bit CAM bank is 0.49nJ. At each cycle, 25 CAM banks for only one chip are activated. So the total energy consumed in CAM banks (bitlines and logic) will be 12.25 nJ per cycle.

2- For 32 bit CCM, the energy consumed per cycle is 24 pJ for all bitlines switching. So the average energy consumed is almost half of this value which is 12 pJ. Since in each cycle 256 of these CCM's activate in one chip, they will consume 3.1nJ of power.

3- From simulation, Priority Encoder was consuming 0.16nJ per cycle. This was for the critical path of the PE. Assuming a factor of 2 for the whole encoder and considering that we have 17 of these PE's on each chip, the overall energy consumption by PE will be 5.4nJ per cycle.

4- Now we should calculate the energy for charging up IP number buses inside the activated chip. From chip floor plan we have 5 of these buses. These buses are 1cm long and so each line has a capacitance of 2pF. So the total capacitance is 320 pF. Since on average half of these lines switch, the energy per cycle will be 1.75 nJ. If a match is found, we only activate one port address bus. Since the port address bus is only 5 bit long, its contribution to energy consumption is negligible.

If we sum up these numbers, the overall energy consumption per cycle (for all chips) will be 22.5nJ per cycle. For 50 MHz clock frequency (20 ns cycle time) this is equivalent to 1.125 W overall or 1125/8≈141mW per chip. For 7ns cycle time (142MHz clock), this power is 400mW per chip.

6.0 Concluding Remarks

When we set out to decide the overall architecture of our design we made the following observations:

- Hardware implementations invariably had poor algorithm design and hence burnt a lot of power
- Software (or processor based) design while having a good algorithmic design suffered from the memory - processor bus bottleneck and were hence serial in nature and also consumed a lot of area but storing inefficient data structure.

Since we started with virgin silicon it did not make sense to take either of these approaches but to come up with a efficient parallel design combining the best of both worlds.

Overall we think we hit all the three key pins: smallest area, lowest power and highest speed without going in any extreme direction. Also, it should be remembered that our hashing is totally flexible without depending on any IP distribution statistics in the address space.

7.0 Generalizations and Extensions

Our hardware-based search and pre-classification ideas are not only limited to the implementation described in this report. Throughout the report we pointed out some generalizations. The following list explains a few more of these generalizations and extensions.

1- We are not limited to SRAM for implementing our CAM and CCM cells. Any kind of memory cell including DRAM can be used as the storage element (circuitry/device).

2- Any kind of matchline implementation can be used for the CAMs. In the current implementation we used series transistors in our CAM matchline (NAND-like matchline). Parallel transistor implementation of the matchline (NOR-like matchline) or any combination of the two might be used as well.

3- In this implementation CCMs were used for doing the 'greater than or equal to' operation. In general CCMs can be used for any comparison operation.

4- The word length for CAMs and CCMs does not have to be 32 bits. The same ideas explained in this report works for any arbitrary word length.

5- The CAM bank size can be chosen arbitrarily (In this implementation it was 100).

6- We had 3 levels of pre-classification in this implementation, out of which 2 of them where in the address space (Fig. 1). The number of levels of pre-classification is not central to our idea and can be chosen as appropriate for the particular application.

7- We are not limited to the single phase clocking scheme, as was used in this implementation.

8- By providing multiple matchlines for each storage element in our CAMs, we can perform several lookups in parallel and further speed up our search.